Using the SDIF Sound Description Interchange Format for Audio Features

Juan Jose Burred, Carmine Emanuele Cella, Geoffroy Peeters, Axel Roebel, Diemo Schwarz

USING THE SDIF SOUND DESCRIPTION INTERCHANGE FORMAT FOR AUDIO FEATURES

Juan José Burred, Carmine Emanuele Cella, Geoffroy Peeters, Axel Röbel and Diemo Schwarz
IRCAM - CNRS STMS
{burred,cella,peeters,roebel,schwarz}@ircam.fr

ABSTRACT

We present a set of extensions to the Sound Description Interchange Format (SDIF) for the purpose of storage and/or transmission of general audio descriptors. The aim is to allow portability and interoperability between the feature extraction module of an audio information retrieval application and the remaining modules, such as training, classification or clustering. A set of techniques addressing the needs of short-time features and temporal modeling over longer windows are proposed, together with the mechanisms that allow further extensions or adaptations by the user. The paper is completed by an overview of the general aspects of SDIF and its practical use by means of a set of existing programming interfaces for, among others, C, C++ and Matlab.

1 INTRODUCTION

The *Sound Description Interchange Format* (SDIF) [1, 2] is an established standard for the precise, well-defined and extensible storage and interchange of a variety of sound descriptions including representations of the signal for analysis/synthesis like spectral, sinusoidal, time-domain, or higher-level models, audio descriptors like loudness or fundamental frequency, markers, labels, and statistical models. SDIF consists of a basic data format framework based on time-tagged frames containing matrices of binary numbers or text, and an extensible set of standard type declarations corresponding to different sound descriptions.

The SDIF standard was created in 1998 in collaboration between Ircam–Centre Pompidou in Paris, France, CNMAT at the University of Berkeley, USA, and the Music Technology Group (MTG) of the Universitat Pompeu Fabra, Barcelona, Spain. It arose out of the need to be able to store and exchange sound representation data between different analysis/synthesis programs, research teams, and institutions, and to enable anyone to interpret the information in the files correctly, needing as little external information as possible. With the previously widely used headerless ASCII formats, this goal was seriously compromised, as well as precision and space efficiency lost.

In the present contribution, we propose a set of extensions to the SDIF description types and conventions and best practices for the application of storing and/or transmitting a wide range of audio descriptors, e.g. as generated by the feature [1] extraction stage of an audio pattern recognition system. In the Audio or Music Information Retrieval fields, this is of interest to assure interoperability between feature extraction and further processing modules, such as feature selection, training, clustering or classification, possibly being developed independently from each other, or even by different institutions. Such a need for standardization can arise in the context of multi-partner evaluation, such as in the Music Information Retrieval Evaluation eXchange (MIREX) [3].

SDIF is an open format, unhampered by licensing or intellectual property restrictions, made by the research community for research and creation. There is a wide range of software available for its use: Libraries in C, C++, Java, Matlab, Perl, and Python, command-line and graphical tools, and plugins for audio-processing systems such as Max/MSP and PureData.

After an introduction to the basics of SDIF (section 2), and a general overview of common different types of audio descriptors (section 3), we will propose several format conventions for storing general descriptor information (section 4). The standard's type declarations are completely extensible and modifiable by the user (section 5), which makes the current proposal adaptable to special application needs. Finally, in section 7, we will present a selection of helpful tools distributed with the SDIF library, programming interfaces for reading and writing SDIF files, a number of applications and online resources.

2 OVERVIEW OF THE SDIF SPECIFICATION

SDIF is a binary meta-format based on streams, frames and matrices. The matrix content representation can be chosen to be text, integer, or floating point in different bit-widths. Several matrices are grouped into a frame, which has a precise 64 bit floating point time tag in seconds relative to the start of the file. Frames are always stored in increasing temporal order and are assigned to a stream. Streams separate different entities of the same type, such as different sound files, channels, or different analyses or descriptors, thus allowing to unite a database of several source sound files, plus various stages of analysis and descriptor data, in one single SDIF file. [2] For example, one could choose to store two channels of audio, their STFT representation, fundamental frequency and sinusoidal partial representation, markers, labels, and harmonicity audio descriptors in one file.

[1] We will use the terms *feature* and *descriptor* interchangeably.

[2] This was formerly achieved by separate files with different file extensions, grouped in the same directory, with the risk of loss of single files, and a certain implicitness of their relationships.

Frame and matrix types are identified by 4-byte ASCII *signatures*, inspired by the IFF format. Each frame starts with a header containing the frame signature, the number of matrices it contains, an integer identifying the stream it belongs to, and the time tag. Each matrix' header contains its signature, its content data type, and its number of rows and columns. Matrices are padded to 8 byte boundaries to achieve 64 bit word alignment for fast access using memory mapped files.

Special header-frames can store global textual meta-data in so-called Name–Value Tables (NVTs), together with relationships of different streams (see section 6). However, global information that is needed to interpret the data, for instance the sampling rate, is stored in information matrices, making SDIF files streamable.

There are a number of standard description types that specify the columns of matrices, and which matrices can occur in which frame type. Examples include `1FQ0` to store fundamental frequencies and `1TRC` to store partial tracks resulting from sinusoidal analysis. By convention, the signatures of standard frame and matrix signatures start with a version number, mostly `1`. It is always possible to declare new experimental description types, or to extend existing types by new matrices for a frame, or new columns for a matrix. New experimental types have signatures starting with `X`. SDIF is designed such that programs can always ignore additional data they don't know how to interpret.

3 OVERVIEW OF AUDIO DESCRIPTORS

Audio descriptors (also named audio features or audio attributes) are numerical values describing the contents of an audio signal according to various points of view, e.g. temporal, spectral or perceptual characteristics. They are used nowadays in many areas such as for the development of query-by-example (search-by-similarity), automatic indexing, segmentation or identification applications. These features are extracted from the audio signal using signal processing algorithms. Several taxonomies can be used to classify them, for example according to the type of content they can be applied to (single musical notes, percussion, sound effects, generic audio, etc.), the underlying signal model they rely on (source-filter, sinusoidal harmonic models, etc.), their abstractness, i.e. what the audio feature represents (spectral envelope, temporal envelope, etc.), their steadiness or variability, i.e., the fact that they represent a value extracted from the signal at a given time, or a parameter from a model of the signal behavior along time (mean, standard deviation, derivative or Markov model of a parameter), or the time extent of the description provided by them: some descriptors apply to only part of the object (e.g., description of the attack of the sound) whereas others apply to the whole signal (e.g., loudness). On the computer side, an audio feature is represented by a numerical value or a vector of numerical values when several audio features can be linked together (such as the coefficients of the MFCC, or various definitions of the spectral centroid).

Each of such numerical values describes a specific property of the signal around a specific time (the time of the segment corresponding to the analysis window). Most audio features are first computed on a small-time scale (often called short-term analysis) which for most of them correspond to the length of the analysis window used for FFT analysis (usually 40 ms to 80 ms). Other features are computed on a longer-time scale due to their semantics (the log-attack-time, the roughness or the fluctuation strength are associated to a whole musical note) or due to computational requirements (the estimation of a vibrato or tremolo at a frequency of 6 Hz necessitates a window larger than 80 ms). In this case the numerical value needs to be associated exactly to the segment used to compute it. Finally, it has become frequent to model the behavior of an audio feature over time. Using long-term analysis (often with window lengths of 500 ms or larger), it is common to compute the mean, variance or derivative values of the feature over the sliding window. This type of description is often named *temporal modeling*, and the corresponding large frames are sometimes called *texture windows* or *macroframes*. In this case, the segment over which the temporal model is computed does not need to have a specific semantic (e.g., "every 200 ms" has no meaning by itself), although it can have (for example for beat-synchronous analysis).

In the proposed SDIF extensions, we address these specific requirements concerning definition of different time horizons, multidimensionality and addition of semantic information. It is out of the scope of this article to present detailed mathematical definitions of the descriptors. A large set of them can be found in [4].

4 EXTENDED TYPES FOR GENERALIZED AUDIO DESCRIPTORS

A defining aspect of SDIF is its flexibility regarding data types. Using the data declaration syntax introduced later in section 5, the user is able to define or redefine the frame/matrix grouping rules and to add or modify type declarations at will. Throughout the paper, we will present different possibilities for different applications in the context of descriptor extraction. At the same time, however, we are proposing a ready-to-use set of type declarations designed according to what we consider appropriate to most audio content analysis and retrieval applications. The proposed type declarations, together with related documentation and several example SDIF files can be accessed online [3] .

In both short-time and temporal modeling cases, the values of a given descriptor at a given time are stored in a separate matrix. For example, spectral centroid values are stored in matrices of signature `1SCN` and for zero-crossing values the matrix signature is `1ZCR`. In analogy with the standard matrix types (e.g., with sinusoidal modeling), the number of rows of the matrix must correspond to the dimensionality of the descriptor, i.e., with the number of frequency bands (such as with spectral flatness) or with the number of coefficients (such as with MFCCs or autocorrelation features).

[3] http://sdif.sourceforge.net/descriptor-types

The columns are intended to store the values for different definitions or implementations of the same descriptor. For example, the first column of the spectral centroid could contain the values for a centroid defined on linear frequency and amplitude scales, and the second the values using logarithmic scalings. We will thus use *dimensions* to denote rows and *variations* to denote columns.

Both the number of rows and of columns are not fixed beforehand by the type declaration. In most cases, the number of rows will depend on a user-defined feature extraction parameter, such as number of frequency bands or cepstral coefficients. It is also most likely that the user will just store a single variation for each descriptor. In some training experiments, however, it can be interesting to store different variations at the same time and let an automatic feature selection algorithm select the most informative one.

Even if not strictly necessary for a clustering or statistical training algorithm, the feature extraction program should store additional information alongside the descriptor values, such as bandwidths, thresholds or labels describing the contents of the columns. This data will be included in information matrices, whose signatures will be equal to the descriptor matrix signatures, with the first character `1` replaced by `I`. Entries on information matrices are always stored column-wise. The next subsections will detail how to store such matrices.

To accommodate the different descriptor time-spans mentioned above (short-, mid-, long-term and temporal modeling) we propose two different schemes for SDIF storage. In fact, we simplify such categorization by only considering, on the one hand, the descriptors themselves (which are directly based on the signal) and, on the other hand, temporal modelings (which are based on descriptors), which will respectively be addressed in the next two subsections. We deliberately avoid a categorical distinction between the length of the time spans, since the only difference from the SDIF point of view is the definition of the time boundaries. Also, we will avoid the terminology "low-level" or "high-level" feature because of the lack of consensus about its meaning. By convention, an SDIF file containing only audio descriptors should end with the double extension `.descr.sdif`.

4.1 Format for Descriptors

We propose the following procedure to store descriptors that are directly based on the signal, independently of their time span. Each data matrix corresponding to a given analysis window of a given descriptor is contained within a frame of type `1DSC`, a *descriptor frame*. In order to be able to perform a one-to-one mapping between streams and descriptors, each descriptor frame must contain only a single descriptor data matrix. The time tag on each frame corresponds to the center of the analysis window. The optional information matrices must be stored at the beginning of the corresponding descriptor stream, and are valid until a new information matrix of the same type appears on the same stream. Figure 1 represents this approach schematically for the case of a file storing spectral centroid (`1SCN`) and perceptual tristimulus (`1PTR`) data. In this example, the spectral centroid is a unidimensional descriptor with a single variation, and two different variations of the three-dimensional tristimulus have been stored, each one defined with a different amplitude scaling.

The SDIF reading routines search first by frames, and then by matrices. Thus, if the performance in accessing individual descriptors is critical, an alternative approach would be to declare a pair of frame type and matrix type with the same signature for each descriptor. Its disadvantage however is that it requires to double the amount of type declarations.

4.2 Format for Temporal Modeling of Descriptors

There are three main reasons for considering temporal modeling of descriptors as a special case that requires dedicated storing conventions. First, they are *derived* features of typically the same dimensionality than their shorter-term counterparts they are based on, and thus they can reuse their matrix type declarations. This results in each temporal model being associated to a frame type, containing matrices of the types already declared for the descriptors. For example, a texture window of the weighted variance of the spectral centroid will correspond to a frame of type `1WVR` containing a matrix of type `1SCN`. An indicative list of types of temporal modeling is: mean, variance, standard deviation, weighted mean, weighted standard deviation, amplitude modulation.

Second, it is possible to assume that their time-span will be relatively long-term (in the range of seconds), and thus performance when accessing them will rarely be critical. If stored together with their short-term descriptors, they will predictably constitute a very small percentage of the file size. This allows more freedom in choosing the most appropriate way of frame and matrix grouping, depending on the application. This can be done either in the one-matrix-per-frame way, as before, or using *compound frames*, i.e. grouping several matrices corresponding to different descriptors under the same frame. In the first case, individual assignment to streams, and different window sizes between different descriptors will be possible. In the compound case, visual inspection of SDIF files (e.g., after ASCII conversion with the **sdiftotext** tool) will be clearer. To leave both possibilities open, the provided type declaration lists all descriptor matrices as optional members of each temporal modeling frame. If visual inspection is not important, we recommend using the one-matrix-per-frame alternative.

Finally, the third reason is that the segment the temporal modeling works on is not necessarily of constant size and overlap. For example, one can define a sequence of log-attack time features resulting from a note-wise segmentation of a melodic line. Thus, a mechanism must be provided to indicate the segment on which the descriptor evolution is modeled. This mechanism makes use of the standard frame type `1MRK` that represents a marker at the time of the frame, containing matrices `1BEG` and/or `1END` with a unique identifier of a segment. Now, the temporal modeling frame will also contain a matrix with the identifier of the segment whose descriptor evolution it models.

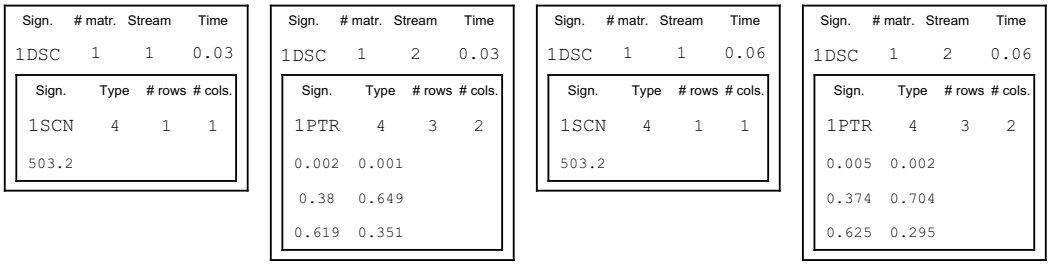

Figure 1. SDIF storage example for four consecutive frames of a short-time spectral centroid (1SCN) and a short-time perceptual tristimulus (1PTR).

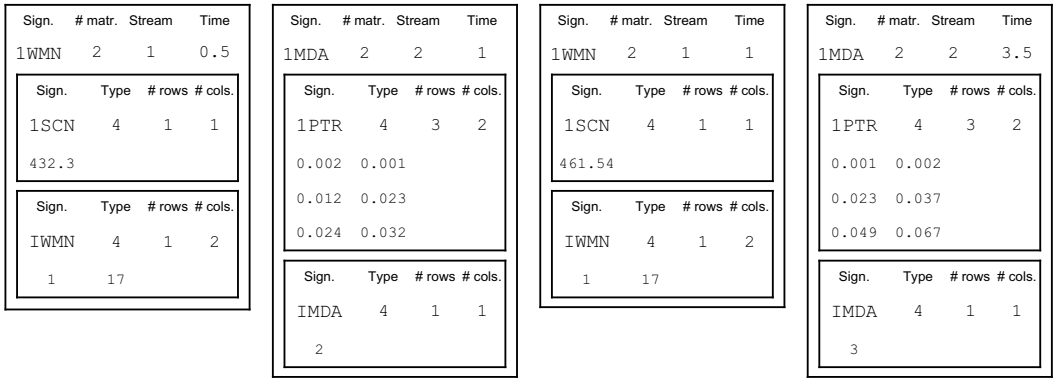

Figure 2. SDIF storage example for four consecutive temporal modeling segments: weighted mean (1WMN) of spectral centroid and modulation amplitude (1MDA) of perceptual tristimulus.

Each temporal modeling frame type has an associated information matrix type, e.g. IWVR for the weighted variance frame type 1WVR. Then, each temporal modeling frame contains one information matrix indicating the duration, in seconds, of the window as its first element. For regularly spaced texture windows, the time tag of the frames corresponds to the middle of the temporal modeling segment. For irregular segments given by markers, the temporal modeling frame is written at the start time of the segment. Figure 2 shows an example of this approach, using single-matrix grouping. It shows the storage for two consecutive texture windows containing the weighted mean (1WMN) of the spectral centroid, and the amplitude modulation (1MDA) of the perceptual tristimulus. Like with any other matrix, the meaning of the columns is contained in the type declaration. As in the plain descriptors case, it is possible to store descriptor-related information matrices at the beginning of the corresponding streams. It should be noted that some temporal modelings, such as log-attack time or temporal increase, are almost always based on a very specific descriptor, in this case the energy envelope. It would be however possible to attach them to no matter which shorter-time feature (for instance, the temporal increase of the fundamental frequency could be an informative "sweep descriptor").

5 USER-DEFINED TYPES

SDIF allows to define new description types by declaring frame and matrix signatures, and matrix columns, or to extend existing types by new matrices, or new columns. The declaration of extended types is included in a special *type declaration* frame of signature 1TYP. Each SDIF file must include a 1TYP frame declaring the non-standard types, and is therefore self-contained. The standard types must not be declared in such a frame, since they are already recognized by the library. The following is an example of the type declaration for some existing types:

```
1TYP
{
  1MTD 1SCN    {SpectralCentroid}
  1MTD 1ZCR    {ZeroCrossingRate}
  1MTD IWMN    {WindowLength}
  1FTD 1WMN
       {
         1SCN  SpectralCentroidWeightedMean;
         1ZCR  SignalZeroCrossingRateWeightedMean;
         1FQ0  FundamentalFrequencyWeightedMean;
         IWMN  WeightedMeanInfo;
       }
}
```

Each line starting with 1MTD is a matrix type declaration, containing the new matrix signature and, between curly

brackets, a list of the declared column names, separated by commas. The lines starting with `1FTD` are frame type declarations; after the frame signature (`1WMN` in the example), between the curly brackets, there is the list of the matrices that can appear inside that frame with their role, terminated by a semicolon. A matrix type must be declared before the frame type containing it. As a side note, we point out that even for standard types (like `1FQ0` in the example), the association with a non-standard frame *must* be created. All the descriptor types proposed here contain only one declared column. This does not hinder the user to store as many additional columns as needed.

6 DECLARATION OF STREAM RELATIONSHIPS

In the previous section we have seen that the proposed SDIF representation of audio descriptors will distribute the data over several streams of SDIF frames. For instance, we could have a stream of temporal modeling frames `1WMN`, modeling segments given by a stream of `1MRK` frames, based on various descriptors in different streams, that were extracted from the audio data in a stream of `1TDS` files.

While the relationships between these streams are clear in our context of use, we would like them to be expressed explicitly, in a machine-readable format, in order to be exploited by tools, visualisers, and other applications. The explicitness of the relationships and meanings of streams becomes even more important when several source sounds, channels, or analyses are included in the same SDIF file.

At the beginning of the SDIF standard, there has been an attempt [5] to formalise the relationships between streams in an XML based language, which has not been followed by concrete implementations. This is possibly due to the comparatively simple SDIF files that were used at that time, posing no real need for an explicit representation, and, what's more, to the problem of adding a dependency on an external library to parse XML to all SDIF tools.

We propose here a new, simple approach how to define relationships, meaning, and content of streams in a tabular text format, not unlike the existing Name–Value Tables. We will start with expressing only those relationships that are indeed needed for the applications described in this article, but we strive to keep the format open enough for other relationships to be added in the future.

The format associates one entity by a relationship to a list of entities. The possible entities are listed in table 1, the list of defined relationships in table 2. For stream IDs and frame and matrix signatures, the notation of entities follows the SDIF-selection syntax [2] to ease understanding and parsing, all other entities are labels, starting with a letter. We propose to store this table in a `2IDS` frame with text matrix.

Entity	Example	Description
# *stream id*	`#23`	Stream number 23
: *frame* / *matrix*	`:1WMN/1SCN`	stream contains weighted mean of spectral centroid
identifier	left-channel	Label

Table 1. List of entities in the proposed stream relationship declaration.

Relationship	Description
contains	left stream contains frame, matrix types given right
group	left label groups labels or streams to the right
segments	left stream contains segmentation information for right entity
derives	left entity is derived (by analysis or temporal modeling) from right entity

Table 2. List of relationships between two entities.

7 SDIF USAGE INFORMATION

7.1 SDIF Tools

In order to easily manipulate the produced SDIF files, the library provides some command-line tools together with the source code. Basically, there are tools to display, to extract and to convert the stored data; the most important are:

• **querysdif**: prints out a summary of the information stored in the file, e.g. the number and the types of stored frames and matrices,

• **sdifextract**: extracts selected data from a file; the output can be stored in many formats like SDIF itself, multi-bpf (text lines of frame time and all selected columns), etc.,

• **sdiftotext, tosdif**: convert data between SDIF and plain text, preserving frame-matrix relationships. These also exist as *droplets* for Macintosh, onto which one can drop a file, generating the complementary text or SDIF file.

7.2 Programming Interfaces

This section will describe the existing possibilities to manipulate SDIF data using the SDIF libraries that are publicly available on the SourceForge software repository (see location below). There exist a C-library (denoted just as the SDIF library) and a C++ library (called Easdif). Easdif incorporates the complete interface of the SDIF library. Additionally Easdif provides a high level programming interface to SDIF files, frames, and matrices including iterators over SDIF files. These iterators allow easy and efficient navigation within the SDIF file using an internal cache of the meta data (frame and matrix headers). The SDIF and Easdif libraries are designed to work on all major platforms (Linux, Mac OS X and Windows), and thanks to the use of the cmake [4] project generator they can be compiled with most common compilers (gcc, MSVC, Intel, . . .).

The Easdif library cannot only be used in C++ applications, it comes as well with a number of bindings for other applications and languages. A recent achievement is the ad-

[4] http://www.cmake.org

dition of sources for Matlab and Octave bindings (mex) that allow access to data stored in SDIF files for these two programing environments. The Matlab/Octave interface makes use of the frame and matrix header cache of Easdif to allow efficient random access to SDIF frames and matrices.

Additionally, the Easdif library includes support for the SWIG [5] wrapper generator and interface descriptions for bindings for python, perl and java. Starting from the existing interface descriptions the addition of other scripting languages that are supported by SWIG should be easy.

7.3 Applications using SDIF

Many applications developed nowadays integrate a general-purpose SDIF reader/writer. Here is a partial list:

• **AudioSculpt** [6] : tool to analyse and modify sounds through a visual approach,
• **ASAnnotation** [7] : sound analysis, visualisation and annotation, based on AudioSculpt,
• **CLAM** [8] : C++ library for music information retrieval,
• **Csound** [9] : language for sound manipulation and analysis,
• **Diphone Studio** [10] : tool for sonic morphing,
• **FTM** [11] for Max/MSP, jMax, PureData: realtime visual programming environment for sound analysis/synthesis,
• **Loris** [12] : sound modeling and processing package,
• **OpenMusic** [13] : visual programming language for computer-aided composition,
• **Open Sound World** [14] : programming environment to process sound in real-time,
• **SDIF-Edit** [15] : SDIF editor and visualisation tool,
• **SPEAR** [16] : sinudoidal analysis/resynthesis.

Moreover, in the developer tools included with the latest version of the Mac OS X operating system (version 10.5, *Leopard*), SDIF support has been included for the AudioUnit called *AdditiveSynth*.

7.4 Availability and Online Resources

SDIF is an open source project hosted on the SourceForge repository. The following resources are available:

• **SDIF Homepage.** (http://sdif.sourceforge.net)
Includes the format specification and main documents, and pointers to all other resources.
• **SDIF Download.** (http://sourceforge.net/projects/sdif)
Includes the sources and several binaries for the SDIF and Easdif libraries. For developers, CVS access is possible.
• **SDIF Wiki.** (http://sdif.wiki.sourceforge.net)
is intended for the documentation of application-specific usages, the proposal of extensions, and additional user-oriented documentation.
• **SDIF mailing lists.** There is one mailing list for users (http://listes.ircam.fr/wws/info/sdif) and one for developers (https://lists.sourceforge.net/lists/listinfo/sdif-devel).

[5] http://www.swig.org
[6] http://forumnet.ircam.fr/691.html
[7] http://www.ircam.fr/anasyn/ASAnnotation
[8] http://clam.iua.upf.edu
[9] http://www.csounds.com
[10] http://forumnet.ircam.fr/703.html
[11] http://ftm.ircam.fr
[12] http://www.cerlsoundgroup.org/Loris/
[13] http://recherche.ircam.fr/equipes/repmus/OpenMusic/
[14] http://osw.sourceforge.net
[15] http://recherche.ircam.fr/equipes/repmus/bresson/sdifedit/sdifedit.html
[16] http://www.klingbeil.com/spear/

8 CONCLUSION

In the context of the growing needs for standardization within the fields of audio-based content analysis and retrieval, we have proposed the use of the well-established and open SDIF format for storing, transmitting and sharing general audio features. We made use of SDIF's extension capabilities to declare a set of extended types addressing the needs of general short-term and temporal modeling descriptors. We are hoping to initiate a discussion among interested researchers, e.g. via the wiki and the mailing lists, to further assess the general needs, and eventually update the type proposals. Concerning future work, we are studying the use of SDIF for other audio-related pattern recognition data, such as feature transformation data and statistical models.

9 ACKNOWLEDGMENTS

This work was supported by the French National Agency of Research (ANR) within the RIAM project *Sample Orchestrator*.

10 REFERENCES

[1] M. Wright, A. Chaudhary, A. Freed, S. Khoury and D. Wessel, "Audio Applications of the Sound Description Interchange Format Standard", *Proc. of the 107th Convention of the Audio Engineering Society (AES)*, New York, USA, 1999.

[2] D. Schwarz and M. Wright, "Extensions and Applications of the SDIF Sound Description Interchange Format", *Proc. of the Int. Computer Music Conference (ICMC)*, Berlin, Germany, 2000.

[3] J. S. Downie, "The Music Information Retrieval Evaluation eXchange (MIREX)", *D-Lib Magazine*, Volume 12, Number 12, 2006.

[4] G. Peeters, "A Large Set of Audio Features for Sound Description (Similarity and Classification) in the CUIDADO Project", *CUIDADO I.S.T. Project Report*, 2004.

[5] M. Wright, A. Chaudhary, A. Freed, S. Khoury, A. Momeni, D. Schwarz, D. Wessel, "An XML-based SDIF Stream Relationships Language", *Proc. of the Int. Computer Music Conference (ICMC)*, Berlin, Germany, 2000.